Final Project: Poetry Generation

McKell Stauffer and Maria Fabiano
CS 501R, Fall 2017
Department of Computer Science
Brigham Young University

Abstract

Throughout history, poetry has played an integral role in the world of art. Across different cultures, it has acted as a form of expression and unity, as well as protest and aesthetics. Poetry also contributes to the study of language. Due to its myriad of diverse forms, it has been an active area of research to be able generate them. In this paper, we attempt to generate random poems using a sequence-to-sequence recurrent neural net (RNN). We explore several variations of our RNN, with the most meaningful and grammatically correct poems coming from a word-RNN which uses the argmax of the outputted probability distribution (that results from a cross-entropy loss) to get the next word.

1 Introduction

Text generation is a very popular topic of research in deep learning. Within the vast amount of text that can be generated, poetry is interesting due to its often strict structure. Past research (please refer to Section 2) has mostly focused on learning a subset of these poem structures. In our paper, we wanted to see what a RNN could learn from being trained on a wide variety of poems with all kinds of structures. We were especially interested to see how the method of choosing the next character influences the effectiveness of the RNN.

The rest of this paper will be organized as follows: Section 2 discusses past research on poetry generation using the RNN structure. Section 3 explores our poem collection processes, pre-processing analysis of the poems retrieved, and a brief exploration of the data set. Section 4 describes our approaches to poetry generation and our measures of success. Lastly, Section 5 displays our results and Section 6 is our conclusion, and also describes future work.

2 Related Work

There have been a variety of different approaches to use a RNN to generate poetry. Examples include using encoders and decoders, augmentations with neural memory, and polishing structures [4] [10] [2]. Many of these approaches involved the use of several different sub-models. For example, some researchers used different structures to generate sub-topics and phrases of the poem given certain keywords. They then passed these sub-topics into an encoder and decoder to generate the poems themselves [5]. Other structures took the generated phrases, clustered them, and iteratively chose from these clusters to create a poem [4].

The most complicated model that we found is in a paper by Zhang-et-al. First, the user supplies keywords. As above, these keywords are passed to an RNN to create phrases to use in the poem. Using these phrases, it uses a convolutional sentence model to generate sentences. After these lines are generated, they are reduced to a content vector using a recurrent context model. The poem is then generated character by character by a final RNN using the above mentioned context vector [4].

Much of the past research has focused on generating Chinese poetry [9]. We assume that this is because Chinese poetry and grammar is very structured, and thus easier for the RNN to learn. For example, in his research, Yi, Li, and Sun successfully built a sequence-to-sequence model to generate quatrains [6]. In fact, in their work, they were able to successfully build a net that was able to keep semantic meaning throughout the poems. As many of these poems dealt with Chinese characters, the majority of the RNNs built poems character by character. One exception to this is a paper by Luo and Huang that generated poems word by word using the mutual information between them. In this paper, we explore both char-by-char and word-by-word generation [3].

The majority of the research papers found their approaches to be successful. The poems that were generated by a conditional variational autoencoder were judged by humans to have been written by humans 45% of the time. As many of these papers have been successful for specific types of poetry or structures, we found it worthwhile to study whether a RNN could generate different types of poetry without being given or trained on a certain poetry paradigm.

Outside of former research, there are several websites that have computer-generated poems. For example, CuratedAI.com has a wide variety of computer generated poems [1]. One blog used the paper by Karpathy on text generation to build a poem generator character by character [7]. Two different models were trained on haikus and limericks, and they were able to successfully generate both types of poems [8].

3 Data

3.1 Data Collection

We scraped from a few websites to obtain nearly 5,000 poems to build our data set. Fortunately, the `robots.txt` file for each website allowed us to scrape. Below is a list of the websites we scraped.

- `poetryfoundation.org`
- `https://www.poets.org`
- `http://www.publicdomainpoems.com`
- `http://poems.poetrysociety.org.uk`
- `https://www.poetryfoundation.org/poems`

This data set required a lot of cleaning. We had to account for the difference between each website, as well as the differences within each website. Some of these websites were very inconsistent (within their own web pages) in how they presented the poems. This complicated how we scraped, as we could not use the same script to scrape even from the same site. It also required a great deal of post-processing. One flaw in our scraper was that it gathered some author biographies in addition to the poems. We had to go through each poem manually to remove these biographies, if they existed. Then, we had to remove all non-ASCII characters and extraneous new line and whitespace characters. As we couldn't read in any files with Unicode characters, we had to replace all of these characters manually as well. We decided to remove all punctuation and convert every word to lowercase (besides the word "I"), so as to reduce the complexity of the vocabulary. Then we removed inappropriate content (also by hand). Lastly, we concatenated all the poems into one large file to feed into our neural net. The whole data gathering and cleaning process took about 10 hours.

3.2 Data Exploration

In the web scraping process, we accidentally grabbed some poems that were not English. In the middle of training, we discovered that our data set includes 25 Spanish poems. We decided to keep them because we had already started training and because we were curious to see how our network would handle the different languages.

This data set contains 1,326,749 total words and 50,066 unique words. Unsurprisingly, the most common words in the data set are transition words and pronouns (see Figure 1 below).

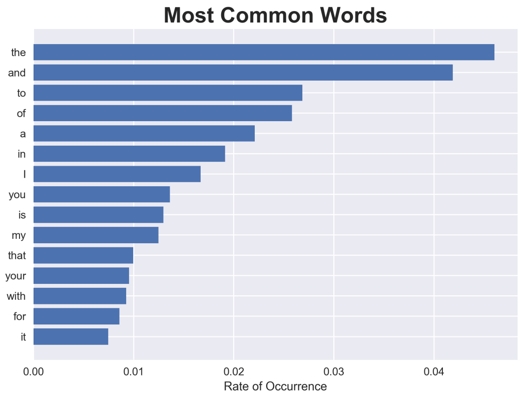

Figure 1: The most common words in the vocabulary.

We also looked at the counts of each character in the poems. See Figure 2.

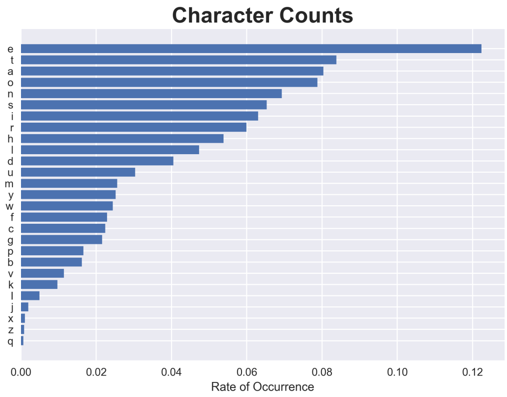

Figure 2: The distribution of the number of words in the poems.

Next, we examined the number of lines in each poem. About 0.28% of the poems in the data set were 200 lines long or more (the longest poem had 705 lines). We omitted those outliers from the Figure 3 (the graph is highly skewed and uninteresting when they are included).

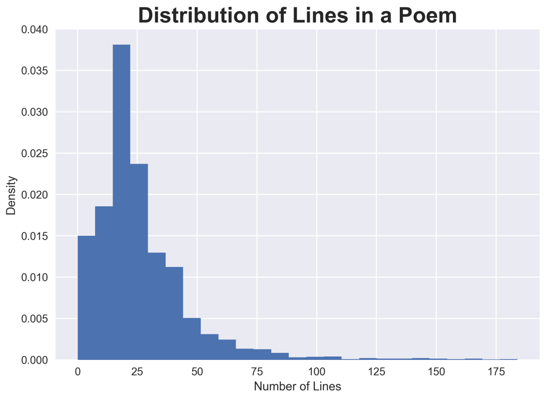

Figure 3: The distribution of the number of lines in the poems.

We did a similar analysis on the number of words in each poem. Again, we omitted outliers from Figure 4, as only 0.22% of the poems contained 2,000 words or more.

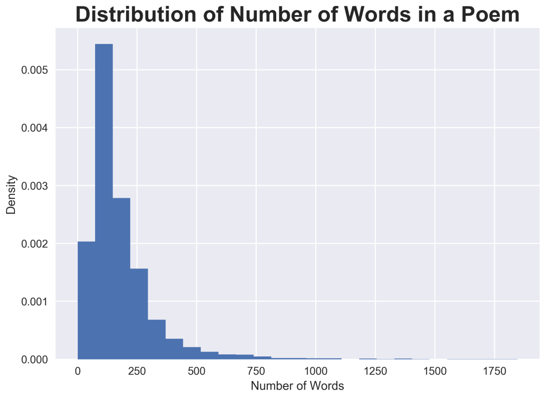

Figure 4: The distribution of the number of words in the poems.

4 Methods

Traditional machine learning methods are generally geared for classification problems. The goal of this project was to essentially create new data; we needed something a little more complex to accomplish this. Deep Neural Networks (DNNs) are very good at generating new data. Consequently, we decided to use a RNN, which is a type of DNN. RNNs have cycles between its units, allowing for a type of internal memory. This memory is useful in text generation to be able to process sequences of inputs and to predict next characters or words based off of previous characters or words. We decided to use a RNN that uses supervised learning by attempting to minimize the cross entropy between the proposed and actual distribution of words. We approached the project this way because we wanted a quantitative measure of success (or failure).

We started our experiments by training on the char-RNN model that we created in class. This model has two layers of LSTM cells. This model generates text character by character, and decides the next character by taking the argmax of the proposed distribution. To calculate loss, it uses the cross-entropy between the proposed and actual distribution of characters. We trained our model with the Adam optimizer (as this is traditionally more effective), and a learning rate of 0.001. This model was our baseline.

From there, we experimented with using a word RNN. Additionally, we changed our model so that it no longer took the argmax of the proposed distribution of words, but instead sampled from it. We also experimented with using temperature to weight the values in the proposed distributions. Lastly, we experimented with using beam search to give our poems more coherence.

We measured the success of our models both qualitatively and quantitatively. For the qualitative measure, we generated poems from our trained model. We do this line by line, with each line having a random word as its prime. At one point, we experimented with using the last word of one line as the prime for the next, but as it did not improve the coherence of our models, we decided to use the random primes. We also specified the number of characters or words that each line should have. Once these poems were generated, we judged them based off of their grammar, coherence, and meaning (as in if they seemed meaningful or not).

We also judged our poems quantitatively with the corresponding losses of the models used to generate them. To compute the loss, we used the cross entropy between the proposed and actual character or word distribution. Overall, we cared more about the qualitative results of our poems than the quantitative. We would rather have meaningful poems produced, than a model that minimizes the loss but produces uninteresting poems.

5 Results

5.1 Baseline: char-RNN with Argmax

For the very first approach to this project, we tried plugging our data set into our char-RNN. While the network learned and produced output, the model was quite repetitive. Most of the output looked something like this: ”the the the the the the. . . ” or ”and and and and and and . . . ”. This is not surprising, especially from the data analysis we did; the most common words in the vocabulary are ”the” and ”and”. After training for a period of time, we did achieve some degree of success. The highest-quality poem we obtained with this approach is below.

Poem 117

true with the world to see you all like a bird
and a man in a dream
and it is a little thing that is not a little

and a good person is made of a love which is not a commodity to be taken away from the path of love
and beauty is like a rose which makes me like a moth
like a rose I am in my heart and soul
my love is my love
you are so beautiful so charming and sweet and chaste my love you are so beautiful so charming and sweet and sweet

However, this poem was generated in the middle of training, and there are clearly some repetitive elements within it. We discovered that, over time, the lines in the poems got shorter and shorter, getting down to just one word per line. The following is an example of one of these truncated poems.

Optimal Loss

another
and
start
and
so
and
start
and
so
and
another
and
start
start
start
start
and shine
and shine
and
so
and shine
and
shine
and

See Figure 5 for an illustration of this model's loss. As seen from this graph, the loss stayed rather constant for the duration of training. The network learned that it could minimize loss by producing short poems, until the point that it was producing one-word poems. It is interesting, though, that while this is what the char-RNN learned, minimizing the length of each sentence did not actually significantly decrease the loss over time (see Figure 5). This model also struggled with using the same words over and over again. For example, the poem above only uses the words "and", "shine", "so", "another" and "start". We found other poems that overly use the words "star" and "street". When we first saw this overuse of words, we decided to take out all punctuation and uppercase letters in our data set to decrease the vocabulary size (and therefore increase each character's rate of occurrence). We found that this did not have any effect on which words were chosen. We hypothesized that this was because we were taking the argmax of the proposed distribution; the network would predict the same characters every time, leading to very little variety. We wanted the network to be a little more robust than this, so we tried other approaches.

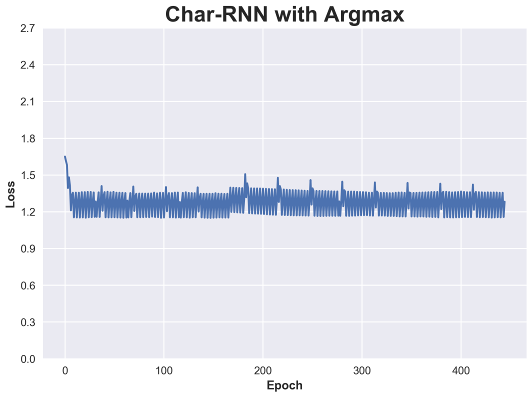

Figure 5: The loss of the model trained with a word vocabulary (using argmax to sample).

5.2 word-RNN with Argmax

For the next network, we used a vocabulary based on words instead of characters. This way, the network would build poems word by word, instead of character by character. We did this in an effort to add more variety to our poems. The only downside to this change is that no new words could be created. As we wanted our poems to be grammatically correct, we had no interest in generating new words anyway. Thus, the network could focus less on making coherent words, but more on putting these words together to create a meaningful poem.

This improved overall performance of the network qualitatively by providing much more varied results. The best poem generated (with a little post-processing in the form of punctuation and line breaks) is as follows:

Poem 199

muses there are.
we are our own timings.
who has to flop and all with care?
friendship is a special day that is given to us in this life's game.
so don't feel bad or mope, just hang in there and try to find a way to be happy,
and show you nice things that're nice and smile,
and smile, and don't feel lost or sad, but don't feel bad or ever cry;
don't feel blue or ever cry or feel shy.

don't feel down or blue or nope,
for you will have to survive and know that you
can do

While this poem is not nearly as repetitive as some of the earlier poems, there are definitely repeated patterns and elements within it. This could be because we were still using the argmax to determine the next word.

See Figure 6 for a visualization of this model's loss. When using a word-RNN, the model took much longer to train, because the vocabulary of words was much larger than the original vocabulary of characters. Thus the epochs in this graph were every 100 poems, whereas for the char-RNN they were every 1000 poems. This could account for the discrepancy between the losses of the word- and char-RNNs. When comparing the two graphs, the word-RNN has a higher loss throughout, but its loss decreases the entire time, whereas the loss for the char-RNN decreased much faster initially, it then periodically increased and then decreased to pretty much the same values.

As we are more interested in the qualitative performance of our models than the quantitative, we found that changing the model to predict the next word instead of the next character was more beneficial.

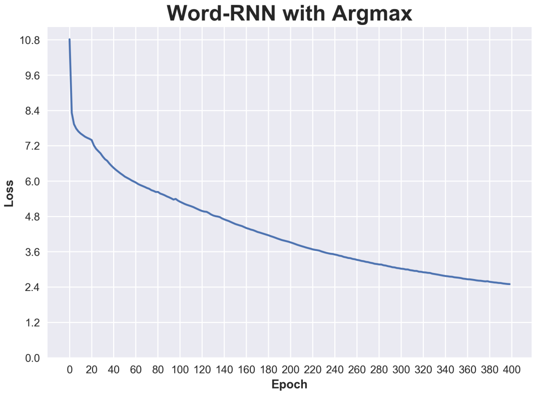

Figure 6: The loss of the model trained with a word vocabulary (using argmax).

5.3 word-RNN with Random Sampling

The next modification we made to the network was to randomly sample the next word from the state distribution, instead of choosing the word with the highest probability (the argmax). With this approach, we expected that the random sampling would prevent the network from getting caught in local minima, thus resulting in a greater variety of words.

Figure 7 visualizes this model's loss.

These losses are nearly identical to the losses using the argmax of the state distribution. We do see that the random sampling method has more frequent peaks (although they are small); this is likely due to the introduction of randomness when the network samples. The final loss for the argmax method was 2.5023, which is slightly higher than the sampling method, which decreased down to 2.4161. Although the loss was slightly smaller, we did not deem it significant enough to say that this model performed quantitatively better than the last model. In addition, we did not deem it to be qualitatively better either. In the end, the poems still had the feeling of gibberish; the random sampling factor was a little too strong. This is perhaps due to the size of the vocabulary; since there are so many words, each individual word has a very low rate of occurrence, as seen in the top 15 most common words in Figure 1. Below is an example of one of the generated poems.

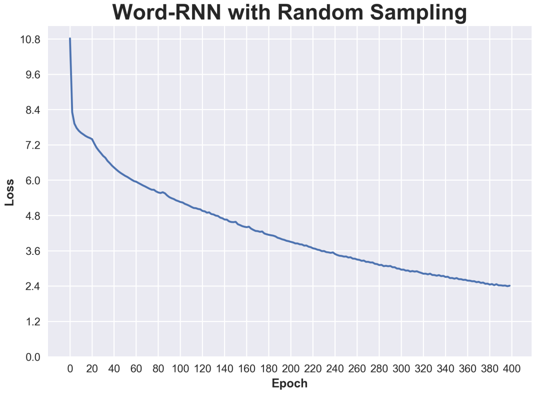

Figure 7: The loss of the model trained with a word vocabulary (using sampling).

Gibberish

the gauge spinning into the dreams with the
power of which might marry their histories are the most oppressed ones on
the lunar cloud
jersey the dust with the wilderness from the
sandy rocks air once more in a medium
since bob contested or the hairline when of
the streets was floating in the burning
basin of men with bodies who fell by
his gums
and he is the day
went clattering faded hearts with her face like
snake walking on what a other comes
one
can feathers appear and balm that drops on a
star falling in far iron till

One of the most interesting types of poems we discovered that our machine generated were "Spanglish" poems. Every once in a while, the network would sample a Spanish word from our data set. However, the network learned that Spanish words are followed by Spanish words. We would occasionally find strings of Spanish in our mostly-English poems. Below is an example.

Spanglish

way para donde unchosen hacia la mis ojos
mas sol pero phone lo algo leaves siempre burning lo en un cochebomba
y como her la esencia de la noche se
himself goddess nobody one and lay his charm against white feet
or when I slipped into its chrysanthemum despair or to plunge though in white amphibian laughter after sighs
down heard its round again days are white note
that time is told for the next mans frenzy word will be picked up
up in great repetition then that's at the others were patient gathering alive as if boy sat down
to came awakes breathing surrounding bourne popped sound pupils pounding atop atop nostalgia steps una falsa okinawa a uno nos combinations en nuestras possessive
le babes regando me
let it be taken over castles on the counter
and makes me free your best in silence till last gleam
my bodies know it much I mean
divine with us in the midst of barren waves
and with humanity indeed to despair and eye without you than some so mistaken this planet is inside

This network was not as effective in coherence when Spanish came along in the poem. It is interesting (and amusing) to note that it did vaguely learn the concept of different languages, even though our data set contained so few Spanish poems.

5.4 word-RNN with Temperature Sampling

We introduced temperature sampling to our model to see what improvements we could make. Temperature is a hyperparameter that affects the softmax function. If we let the temperature be τ, we divide the logits by τ before applying the softmax function. The larger τ is, the more confident the network will be in the predicted states, but those states will be less varied and random. That is, as τ approaches 1, the sampling behaves more like the argmax method. We tried temperature sampling for two values of τ. We used $\tau = 0.7$ and $\tau = 0.1$. The results were disappointing. For even the smaller value of τ, the network was quite repetitive; little variation was introduced into the output. We will not include an example poem in this paper, as the poems were fairly trivial and repetitive (for example, many of the poems were ”the the the the the...”). Changing the value of τ did not change the quality of the poems being generated.

Figure 7 visualizes this model's loss (with $\tau = 0.1$).

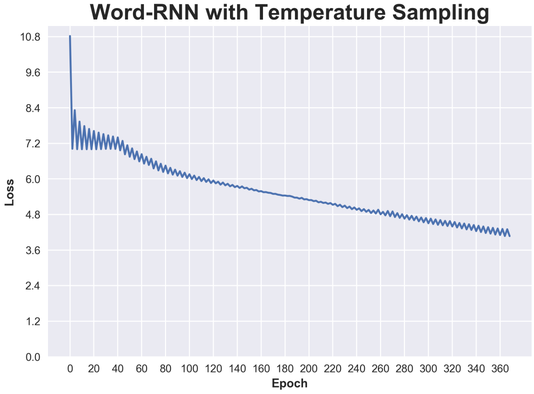

Figure 8: The loss of the model trained with a word vocabulary (using temperature sampling).

5.5 Beam Search

We wanted to implement a beam search in our network to increase its long-term performance. We thought that this would allow for more coherence throughout poem, leading to poems with more meaning. However, we were unsuccessful in applying it to our RNN. We spent several hours running into Tensorflow errors with our implementation. We found many of the prebuilt Tensorflow functions to be unhelpful. For example, `tf.nn.ctc_beam_search_decoder()` did not work in our implementation as it attempted to decode what we had already decoded. We could not find a function in Tensorflow that did not involve decoding in its beam search. We then tried to implement it by hand, but did not ultimately succeed.

5.6 GRU Cells

The next modification we made in the network was changing the LSTM cells to GRU cells. We were not able to train as long with this model due to time constraints, but we noticed an interesting pattern in the loss (see Figure 9). After a period of stagnation, the model seemed to hop out of a local minimum and keep learning. While the loss decreased at a similar rate as previous models, the poems were still quite random and gibberish-sounding.

Below is a poem generated by this model.

Poem 102

crawl settles
blue gold is a million instrument, all sets from mountains
moving from one snaking tube leaving behind a secondhand wishbone
troy in contempt shapes hatred; faith burns
renew us powerful chicksits, you are not here
is made in nothing
tangle of sourdough creamed, windy dried rocks, the crushed rocks

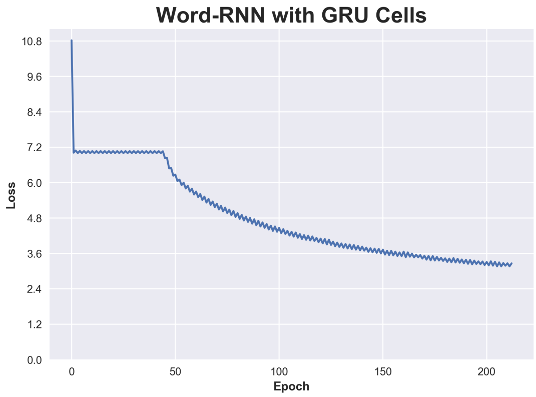

Figure 9: The loss of the model with GRU cells.

5.7 Four Layers

The last modification we made to the network was adding more LSTM cells. We added two more cells to have a total of four LSTM cells. The loss for this model is visualized in Figure 10. The loss did not decrease as much as it did for other models (we were not able to train this network for as long, either). The loss did not decrease as greatly as the loss of previous networks.

A generated poem from this network is below.

Poem 82

stone step with a luminous rosy dark building
and made court limestone thousands with love and phases to light that loving
heard green and broken repent roasted wise, all kind smelling five illuminated better jumps you detest when my sighs
we turn in; all the conditional ones flew

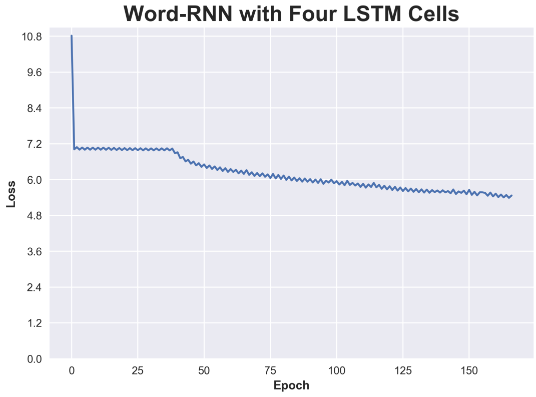

Figure 10: The loss of the model with four LSTM cells.

6 Conclusion

In this paper, we explored several different implementations of RNNs in an effort to generate poetry. We found that the best of these variations used word-to-word encoding with an argmax over the state distribution to sample. Although the poems made with the argmax struggled with repetition, we found that not using the argmax lead to poems that were overly random. Although the poems generated by these two models differed qualitatively, they performed similarly on the minimization of the loss. We also found that using temperature to try to reduce the randomness of sampling from the state distribution did not work well.

Along the way, we also learned that for a char-RNN, minimizing cross-entropy leads to very short sentences with very short words. In this case, it seemed that training for a long time marginally helped the generator quantitatively, but did hurt it qualitatively. We also found that this structure could learn different languages when trained on both at the same time. When a Spanish word was introduced, often the poem would continue in Spanish for a few words or phrases.

Overall, we found that, as has previously been shown, poems can be generated using recurrent neural networks. What was unique to our results was that RNNs could learn on a variety of different poems (not just Chinese quatrains). Additionally, we found that they are also capable of producing poetry without using many other mechanisms to build the poem. For example, we did not give it a certain structure to follow, nor did we have to generate sub-topics or sentences for it to chose from.

6.1 Future Work

To further improve the network's performance, we have several ideas. First, it would be helpful to look at using different loss functions. We used only cross-entropy, which seemed to struggle with minimizing the number of characters per line instead of creating more probable and coherent lines. We are especially interested in looking at the negative log likelihood, KL-divergence, Poisson loss, and cosine-proximity loss. For understanding, it would also be good in future work to figure out why minimizing the number of characters per line minimizes the cross-entropy.

We are also interested in tweaking the hyperparameters and basic structures of our network. For example, we could continue to change the number of layers (making it much deeper) and the learning rate. Additionally, we could try to change the type of optimizer. Although Adam is generally a good optimizer, we could try to use one with momentum in an attempt to avoid local minima.

In our study, we found it hard to decrease the repetition of words in a poem without introducing too much randomness. We need to find a way to skew the distribution so that randomly sampling from it gives values closer to the most probabilistic value. In other words, we need to find a way to weight the probabilities so that more probabilistic characters or words are much more likely to be chosen than less probabilistic ones. Temperature was supposed to do this, but it was too powerful and made poems that were even more repetitive than the argmax. It would be interesting to look into why temperature did not work on our distributions. It could be that the value of τ was too low or too high, so varying the value of τ could give better results. Additionally, it would be good to look for and use other ways to decrease the repetition in poems without increasing the randomness of the poems by too much.

Greater coherence in generated poetry is very important for future work. One way to achieve this would be to train the network around a central topic. Additionally, we could gain greater coherence sentence by sentence. Getting beam search to work could help with creating more coherent sentences. Also, to have the sentences be better correlated, we could work more with our primes. Instead of having random primes for each sentence, we could again experiment with using the last word of the previous sentence as the prime for the next sentence. This did not seem to help much on the char-RNN, but as the char-RNN did not work as well anyway, it could help on the word-RNN. Additionally, we could decide a topic before hand and find a way to implement this topic throughout.

Lastly, it would be good to put back in punctuation and capital letters (as this would make poems more robust). This would be interesting for our word-RNN, as some punctuation would have to be counted as its own word (such as periods, commas, semicolons, etc.), while other punctuation should be part of a word (such as apostrophes, hyphens, etc.). Additionally, it would be good to add in end-token characters. That way, the RNN could decide for itself how long the poems are. This addition could come with the complication of the minimization of the loss from making shorter poems.

7 Last Words

To conclude this paper, and really showcase the ”deep” learning side of this project, we leave you with some wise words from our poetry generator (with some punctuation added).

Robot Wisdom

I was born to be eternalized and therefore loved.
way to be in the process my sweetheart, my beloved, my love,
my love, my love bring.
I'll get together to be a source of hope and peace, and bring you a brighter view;

and know that you can see the path of joy and smile and learn to live in the way,
and learn to live with joy, and smile, and know that you can learn to live in the way, and learn to live with joy, and smile,
and know that you can learn to live in the way, and learn to live with joy, and smile.

References

[1] Curatedai. `http://curatedai.com/categories/#poetry`. Accessed: 2017-12-19.

[2] *i, Poet: Automatic Poetry Composition through Recurrent Neural Networks with Iterative Polishing Schema*, Twenty-Fifth International Joint Conference on Artificial Intelligence.

[3] *Text steganography with high embedding rate: Using networks to generate Chinese classic poetry*, ACM workshop on Information Hiding and Multimedia Security. Association for Computing Machinery, Inc.

[4] *Chinese Poetry Generation with Recurrent Neural Networks*, Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.

[5] *Can Machine Generate Traditional Chinese Poetry? A Feigenbaum Test*, Lecture Notes in Computer Science, 2016.

[6] *Generating Chinese Classical Poems with RNN Encoder-Decoder*, Lecture Notes in Computer Science, 2017.

[7] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Accessed: 2017-12-19.

[8] sballas. `http://sballas8.github.io/2015/08/11/Poet-RNN.html`. Accessed: 2017-12-19.

[9] Z. Wang, W. He, H. Wu, H. Wu, W. Li, H. Wang, and E Chen. Chinese poetry generation with planning based neural network. 2016.

[10] J. Zhang, Y. Feng, D. Wang, Y. Wang, A. Abel, S. Zhang, and A. Zhang. Flexible and creative chinese poetry generation using neural memory. 2017.